PL-TR-
97-1148

HEALTH INFORMATION INFRASTRUCTURE

Alex Quilici

University of Hawaii at Manoa
Department of Electrical Engineering
Holmes Hall 483
2540 Dole Street
Honolulu, HI 96822

March 1997

Final Report

19980116 119

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION IS UNLIMITED.

AIR FORCE RESEARCH LABORATORY
Space Vehicles Directorate
3550 Aberdeen Ave SE
AIR FORCE MATERIEL COMMAND
KIRTLAND AIR FORCE BASE, NM 87117-5776

DTIC QUALITY INSPECTED 3

PL-TR-97-1148

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data, does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report has been reviewed by the Public Affairs Office and is releasable to the National Technical Information Service (NTIS). At NTIS, it will be available to the general public, including foreign nationals.

If you change your address, wish to be removed from this mailing list, or your organization no longer employs the addressee, please notify AFRL/VSS, 3550 Aberdeen Ave SE, Kirtland AFB, NM 87117-5776.

Do not return copies of this report unless contractual obligations or notice on a specific document requires its return.

This report has been approved for publication.

RICHARD DEJONCKHEERE, Ph.D.
Project Manager

FOR THE COMMANDER

L. KEVIN SLIMAK, GM-15
Chief, Surveillance & Control Division

BRUCE A. THIEMAN, COL, USAF
Deputy Director, Space Vehicles

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

Public reporting burden for this collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington, VA 22202-4302, and to the Office of Management and Budget, Paperwork Reduction Project (0704-0188), Washington, DC 20503.

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE March 1997	3. REPORT TYPE AND DATES COVERED Final; 3/95 - 3/97

4. TITLE AND SUBTITLE
Health Information Infrastructure

5. FUNDING NUMBERS
C: F29601-95-K-0014
PE: 61101E
PR: ARPA
TA: TC
WU: AC

6. AUTHOR(S)
Alex Quilici

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Hawaii at Manoa
Department of Electrical Engineering
Holmes Hall 483
2540 Dole Street
Honolulu, HI 96822

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

AFRL
3550 Aberdeen Ave. SE
Kirtland AFB, NM 87117-5776

10. SPONSORING/MONITORING AGENCY REPORT NUMBER

PL-TR-97-1148

11. SUPPLEMENTARY NOTES

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for Public Release; Distribution is Unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT *(Maximum 200 Words)*

Users often work directly with a collection of legacy simulation programs. These users are responsible for producing the input files for these programs, executing them, and managing their results. This project is to design and construct a general-purpose environment to support this process. In particular, we explore how to base this environment on an explicit object-oriented domain model that describes the domain actions that are simulated by existing simulation programs and the domain objects that are provided as input to or generated as output from these programs. The goal is to demonstrate that it is both possible and beneficial to construct an environment through which users interact with legacy simulation programs solely through an explicit domain model. This report describes the general architecture of such an environment and provides detailed examples that show how this environment can be applied to support users working with a collection of programs that simulate the formation and orbit of space debris.

14. SUBJECT TERMS

Knowledge-Based Environment, Legacy Simulations, Object-Oriented, Domain Model, Simulation Execution

15. NUMBER OF PAGES
84

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
Unclassified	Unclassified	Unclassified	Unlimited

NSN 7540-01-280-5500

Dr. deJonckheere
(505) 846-5054

Standard Form 298 (Rev. 2-89)
Prescribed by ANSI Std. 239-18
28-102

Executive Summary

Users often work directly with a collection of legacy simulation programs. These users are responsible for producing the input files for these programs, executing them, and managing their results. This project is to design and construct a general-purpose environment to support these tasks. In particular, we explore how to base this environment on an explicit object-oriented domain model that describes the domain actions that are simulated by existing simulation programs and the domain objects that are provided as input to or generated as output from these programs. The goal is to demonstrate that it is both possible and beneficial to construct an environment through which users interact with legacy simulation programs solely through an explicit domain model. Our initial target users are those working with a collection of programs that simulate the formation and orbit of space debris.

Background

The US Air Force has a large variety of legacy simulation programs. These programs often share a set of characteristics that make them difficult to use; namely, they can perform multiple tasks, take multiple input files with complex input formats and constraints, and produce multiple output files. These programs are useful, but exceedingly difficult to use. Users wind up constructing complex input files by hand and maintaining large directories of simulation inputs and outputs.

It is possible to make these programs significantly easier to use by wrapping them in a nice visual interface, customized to each program. However, besides being expensive, doing so fails to provide support for managing simulation results, integrating the inputs and outputs of different legacy simulations, and using simulations to perform higher-level tasks, such as parametric studies.

Objective

The objective of this work is to provide an environment that addresses these problems. Specifically, our target is an environment that provides a single interface that is suitable across a wide collection of different legacy simulation programs, where this interface supports management of simulation results and integration of different simulations, while requiring only a formal description of each simulation's inputs, outputs, and performed function.

Approach

Our approach is to describe each simulation program in terms of an object-oriented domain model. Each program corresponds to a set of domain actions; each action requires a set of domain objects as input and produces a set of domain objects as output. The environment then supports acquisition of input objects from the user, mapping of domain objects to input files, execution of programs on those files, mapping of the output files to additional domain objects, and storage and retrieval of domain objects.

DTIC QUALITY INSPECTED 3

Technical Challenges

One challenge is inherent in producing any general-purpose environment: it is necessary to provide an overall architecture for the system, which requires determining what the components of the environment are and how they should communicate with one another.

Another challenge is specific to constructing our particular environment: it is necessary to determine exactly what new functionality our object-oriented domain model buys us – beyond making it easier to execute legacy simulation programs. That is, we must determine how such an environment can best support the management of simulation results and the integration of different simulation programs.

The final challenge is specific to determining whether our approach can work at all. This involves taking a sample domain and trying to create a domain model that covers the set of simulation programs in that domain, as well as forming the mappings between the domain model and the actual inputs and outputs of those programs.

Results

This project has resulted in three things: a specification of the overall architecture for this environment, a determination of the functionality that is possible given our approach and how it can be implemented, and an actual domain model for a small collection of programs that simulate properties of space debris. A shortcoming of the project is that the entire environment has not been implemented, only portions of it to test some key ideas.

Payoffs

The potential payoffs of this approach are that it may well significantly lessen the time it takes for users to perform needed simulation-based studies, such as parametric studies. This potential payoff arises because this approach hides the details of the simulation programs from their users and provides significant support for the management and use of simulation results. It also arises because it changes the problem of generating a convenient interface to legacy simulation programs from coding a program-specific interface into providing a program-specific description of its inputs, outputs, and function – a task that appears to be much easier. In addition, many issues that can't currently be conveniently addressed (such as dealing with large sets of simulation results) are handled automatically, without doing anything extra for a given legacy simulation program.

Recommendations

One recommendation is to continue this work, focusing on several tasks. The first task is to complete a robust implementation of the entire environment, or at least a complete subset of it. The second is to then experiment with this environment using our current domain model. This should give a quick idea of how much of a payoff there is with this environment. Assuming that the environment appears useful, the final recommendation is to form a domain model for a larger set of AF simulation programs in a domain other than space debris simulation, and to determine how well this environment and approach supports execution of this idea.

Overview Of Our Initial Approach

Jump To: [Next Page]

Our initial approach (Phase One) focuses on dealing with individual simulation programs in isolation.

We provide an environment that allows users to execute simulations in the following way:

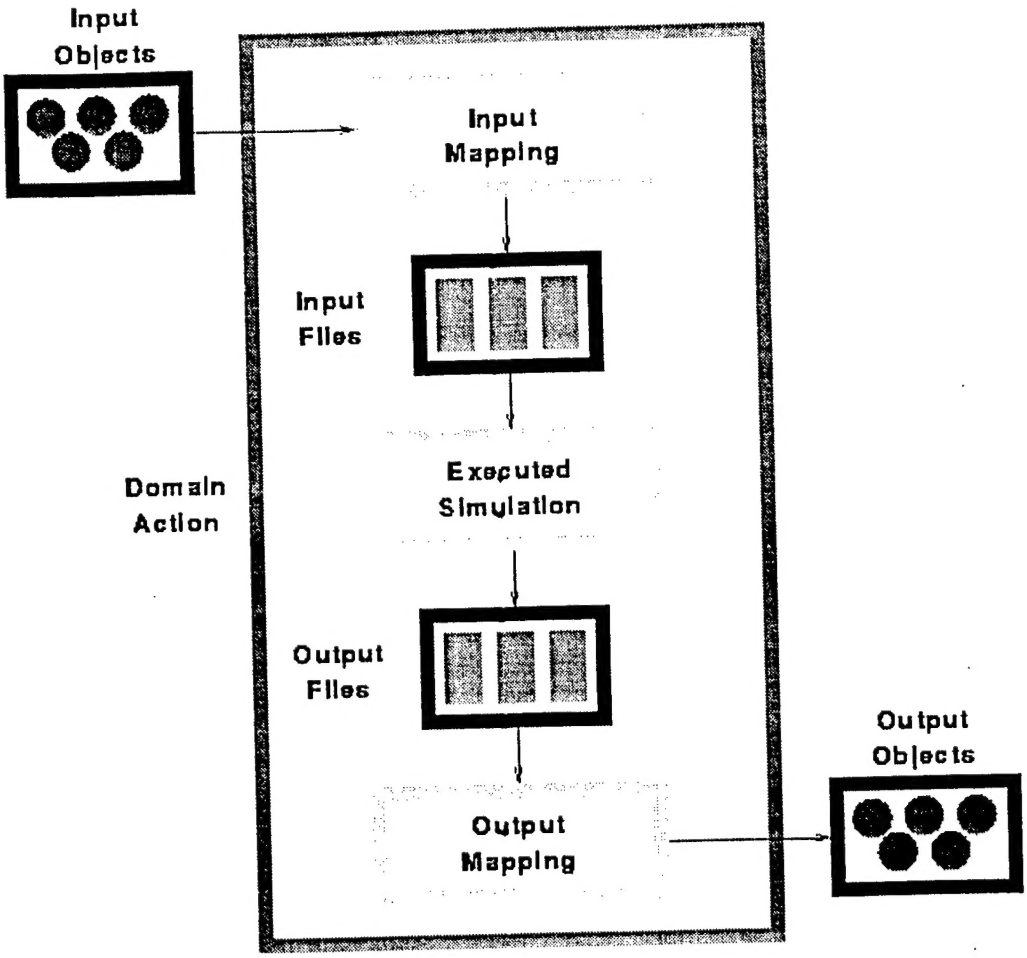
- The user selects a domain action to simulate.
- The user provides the domain action's input parameters, specified in terms of domain objects and attributes.
- The parameters are automatically mapped to input files.
- The underlying simulation program or script is executed.
- The resulting output files are mapped to a set of domain objects.
- The user can then examine these objects or apply tools to visualize them.

The environment (not the individual simulation programs) handles all interaction with the user and all management of simulation results.

[UH/ISE Home Page | Phase One Report Index]

How Simulations Are Executed

Jump To: [Previous Page | Next Page]



[UH/ISE Home Page | Phase One Report Index]

The Contents Of The Domain Model

Jump To: [Previous Page | Next Page]

The domain model necessary to describe exactly what a simulation program does must describe the different tasks the program can simulate, their input parameters, and their expected results.

For example, consider *IMPACT*, a program that simulates space-based breakup events, such as collisions and explosions.

The domain model necessary to describe what IMPACT does must do two things:

- Describe the different tasks IMPACT can simulate, such as various explosions and collisions.
- Describe the input parameters and expected results of these tasks in terms of domain objects, such as satellites, PBVs, boosters, debris clouds, debris fragments, and so on.

[UH/ISE Home Page | Phase One Report Index]

An Example Domain Model: IMPACT

Jump To: [Previous Page | Next Page]

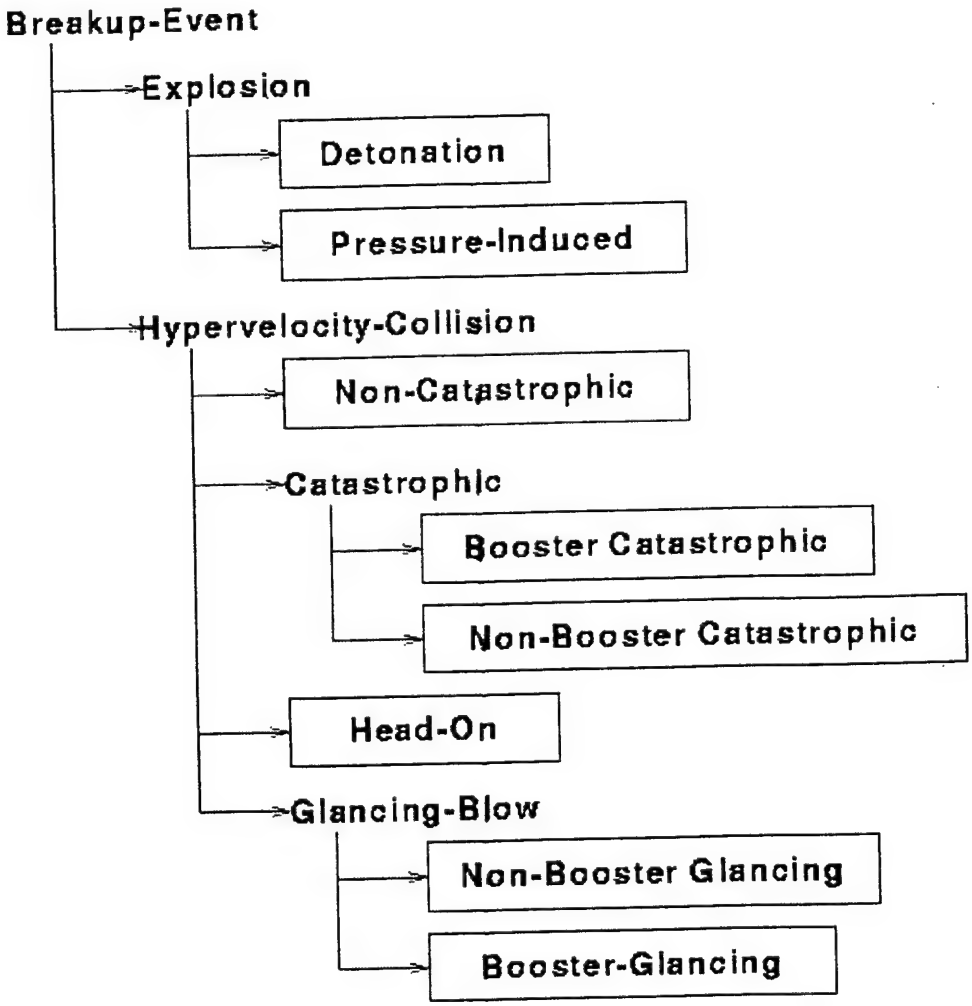
Below are links to a pictorial view of the tasks IMPACT can simulate, as well as to an object-oriented domain model that captures IMPACT's simulation of these tasks.

- Example Domain Model: IMPACT Simulated Events
- Example Domain Model: IMPACT And Explosions
- Example Domain Model: IMPACT And Collisions
- Example Domain Model: IMPACT And Space Entities
- Example Domain Model: IMPACT And Space Debris
- Example Domain Model: IMPACT And Time
- Example Domain Model: IMPACT And Orbital Location
- Example Domain Model: IMPACT And Unit Values

[UH/ISE Home Page | Phase One Report Index]

Example Domain Model: IMPACT Simulated Events

Jump To: [Next Page]



[UH/ISE Home Page | Phase One Report Index | Domain Model Example - Main Page]

Example Domain Model: IMPACT And Explosions

Jump To: [Previous Page | Next Page]

```
Breakup-Event
  |   INPUT:
  |     Epoch [Absolute-Time]
  |     Min-Fragment [Length]
  |
  +-- Explosion
        |   INPUT:
        |     Target [Tank-Containing-Entity]
        |     Available-Energy [Energy]
        |   OUTPUT:
        |     Fragments [Debris-Cloud]
        |
        +-- Detonation
        |         INPUT:
        |           Target (*) [Detonatable-Tank-Containing-Vehicle]
        |           %-Energy-To-Heat [Unitless-Value]
        |           %-Energy-To-Spreading [Unitless-Value]
        |
        +-- P.I. Explosion
                  INPUT:
                    Target (*) [PI-Explodable-Tank-Containing-Vehicle]
                    Time-To-Act [Relative-Time]
                    Pressure-Build-Up [Pressure]
```

[UH/ISE Home Page | Phase One Report Index | Domain Model Example - Main Page]

Example Domain Model: IMPACT And Collisions

Jump To: [Previous Page | Next Page]

```
Breakup-Event
  |   INPUT:
  |     Epoch [Absolute-Time]
  |     Min-Fragment [Length]
  |
  +-- Hypervelocity-Collision
        |   INPUT:
        |     Target [Space-Entity]
        |     Projectile [Space-Entity]
        |   OUTPUT:
        |     Mass-Transfer [Mass]
        |     Relative-Velocity [Velocity]
        |     Number-Of-Debris-Clouds [Unitless-Value]
        |     Debris-Clouds [Collection of Debris-Cloud]
        |     Fragment-Mass-Bin [Mass-Bin]
        |
        +-- Non-Catastrophic
        |
        +-- Head-On
        |
        +-- Catastrophic
        |     |
        |     +-- Booster-Catastrophic
        |     |         INPUT:
        |     |           Target (*) [Booster]
        |     |
        |     +-- Non-Booster-Catastrophic
        |               INPUT:
        |                 Target (*) [Non-Booster]
        |
        +-- Glancing-Blow
              |   INPUT:
              |     %-Directly-Fragmented [Unitless-Value]
              |
              +-- Booster-Glancing-Blow
              |
              +-- Non-Booster-Glancing-Blow
```

[UH/ISE Home Page | Phase One Report Index | Domain Model Example - Main Page]

Example Domain Model: IMPACT And Space-Entities

Jump To: [Previous Page | Next Page]

```
Space-Entity
  |   HAS:
  |     Position [Orbital-Location]
  |
  +-- Tank-Containing-Entity
  |     |    HAS:
  |     |       Vehicle-Type [VEHICLE-LABEL: Booster, PBV, Satellite]
  |     |
  |     +-- Tank-Containing-Vehicle
  |           |   HAS:
  |           |      Tank [Tank]
  |           |
  |           +-- Detonatable-Tank-Containing-Vehicle
  |           |         HAS:
  |           |           Tank (*) [Detonatable-Tank]
  |           |
  |           +-- PI-Explodable-Tank-Containing-Vehicle
  |                     HAS:
  |                       Tank (*) [PI-Explodable-Tank]
  |
  +-- Fragment
        HAS:
          Spread-Velocity [3D-Velocity]

Space-Entity-Component
  |
  +-- Tank
        |
        +-- Detonatable-Tank
        |     HAS:
        |       Dry-Weight [Mass]
        |       Material-Density [Density]
        |       Thickness [Length]
        |
        +-- PI-Explodable-Tank
              HAS:
                Shape [SHAPE-LABEL: spherical, cylindrical]
```

[UH/ISE Home Page | Phase One Report Index | Domain Model Example - Main Page]

Example Domain Model: IMPACT And Space Debris

Jump To: [Previous Page | Next Page]

```
Space-Debris
 |
 +-- Debris-Cloud
 |       HAS:
 |         Post-Collision-Velocity [3D-Velocity]
 |         KE-To-Heat [Energy]
 |         KE-To-Spread [Energy]
 |         Fragment-Spread-Velocities [Collection of Fragment-SV-Group]
 |         Bin-With-Largest [Unitless-Value]
 |         Bin-With-Smallest [Unitless-Value]
 |         Num-Fragments [Unitless-Value]
 |         Min-Fragment-Size [Length]
 |         Fragments [Collection of Fragment]
 |
 +-- Fragment-Group
 |       HAS:
 |         Bin-Containing-Group [Unitless-Value]
 |         Characteristic-Velocity [Velocity]
 |         Max-Possible-Spread-Velocity [Velocity]
 |         Num-Fragments [Unitless-Value]
 |         Num-Spread-Velocities [Unitless-Value]
 |         Entries [Collection of Spread-Velocity-Group]
 |
 +-- Spread-Velocity-Group
 |       HAS:
 |         Num-Fragments [Unitless-Value]
 |         Spread-Velocity [Velocity]
 |
 +-- Mass-Bin
 |       HAS:
 |         Num-Bins [Unitless-Value]
 |         Entries [Collection of Mass-Bin-Entry]
 |
 +-- Mass-Bin-Entry
 |       HAS:
 |         Entry-Index [Unitless-Value]
 |         Entry-Max-Mass [Mass]
 |         Entry-Max-Size [Length]
 |         Entry-Min-Size [Length]
 |
 +-- Fragment
         HAS:
           Entry-Index [Unitless-Value]
           Velocity [3D-Velocity]
```

[UH/ISE Home Page | Phase One Report Index | Domain Model Example - Main Page]

Example Domain Model: IMPACT And Time

Jump To: [Previous Page | Next Page]

```
Clock-Info
 |
 +-- Absolute-Date
 |       HAS:
 |         Month [Unitless-Value]
 |         Day [Unitless-Value]
 |         Year [Unitless-Value]
 |
 +-- Time-Info
      |
      +-- Absolute-Time
      |       HAS:
      |          Date [Absolute-Date]
      |          Time-Within-Day [Relative-Time]
      |
      +-- Relative-Time
              HAS:
                 Hours [Elapsed-Time | Time-Unit=hours]
                 Minutes [Elapsed-Time | Time-Unit=minutes]
                 Seconds [Elapsed-Time | Time-Unit=seconds]
```

[UH/ISE Home Page | Phase One Report Index | Domain Model Example - Main Page]

Example Domain Model: IMPACT And Orbital Location

Jump To: [Previous Page | Next Page]

```
Location-Info
 |
 +-- Orbital-Location
 |       HAS:
 |          Reference-Frame [REF-LABEL: earth-centered, spherical, classical]
 |          Actual-Position [3D-Position]
 |          Actual-Velocity [3D-Velocity]
 |
 +-- 3D-Position
 |       HAS:
 |          X [Length]
 |          Y [Length]
 |          Z [Length]
 |
 +-- 3D-Velocity
         HAS:
            Xv [Velocity]
            Yv [Velocity]
            Zv [Velocity]
```

[UH/ISE Home Page | Phase One Report Index | Domain Model Example - Main Page]

Example Domain Model: IMPACT And Unit Values

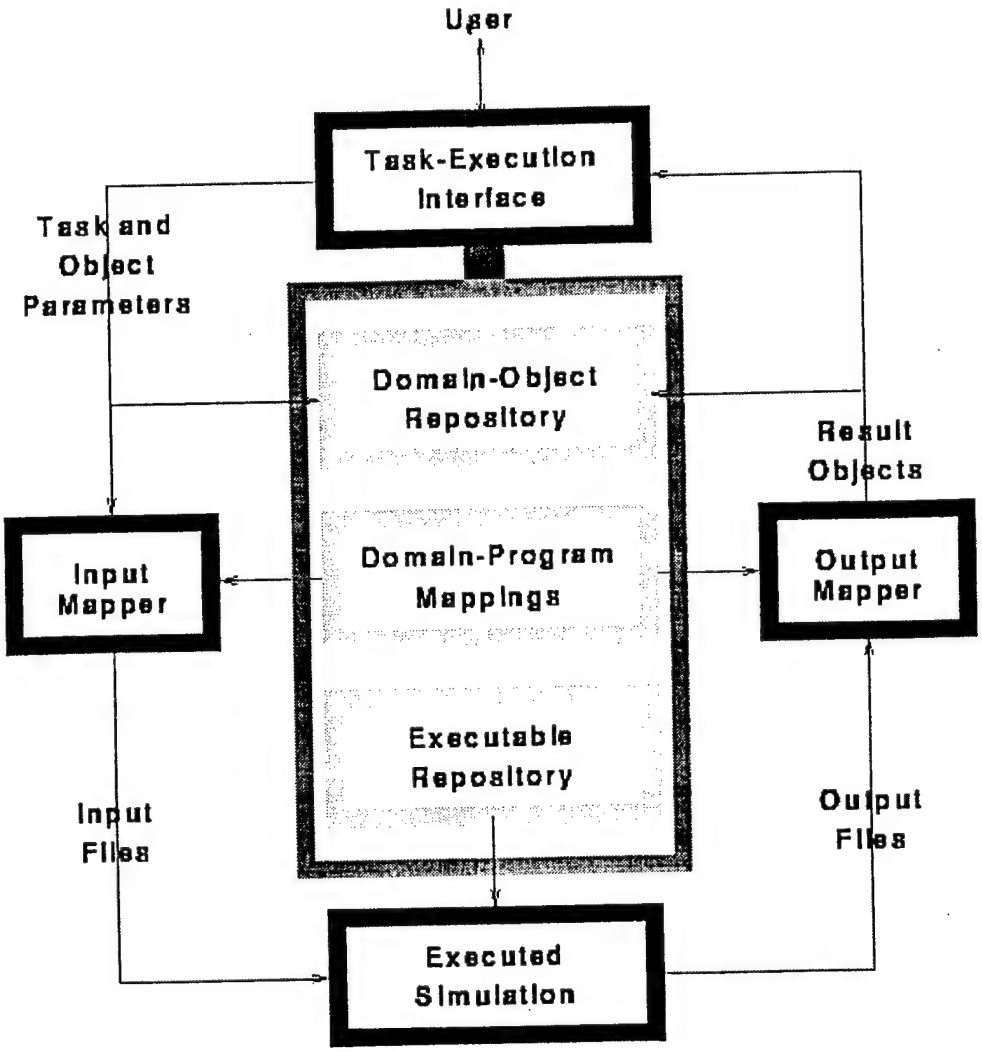
Jump To: [Previous Page]

```
Values
  |
  +-- Sole-Value
  |     |   HAS:
  |     |     Actual-Value [Unitless-Value]
  |     |
  |     +-- UnitLess-Value
  |     |
  |     +-- Unit-Based-Value
  |           |
  |           +-- Elapsed-Time
  |           |     HAS:
  |           |       Time-Unit [TIME-LABEL: seconds, minutes, etc...]
  |           |
  |           +-- Length
  |           |     HAS:
  |           |       Length-Unit [LENGTH-LABEL: feet, meters, miles, etc...]
  |           |
  |           +-- Mass
  |           |     HAS:
  |           |       Mass-Unit [MASS-LABEL: lbs, g, kg, etc...]
  |           |
  |           +-- Velocity
  |           |     HAS:
  |           |       Length-Unit [LENGTH-LABEL]
  |           |       Time-Unit [TIME-LABEL]
  |           |
  |           +-- Density
  |           |     HAS:
  |           |       Mass-Unit [MASS-LABEL]
  |           |       Volume-Unit [VOLUME-LABEL]
  |           |
  |           +-- Pressure
  |                 HAS:
  |                   Mass-Unit [MASS-LABEL]
  |                   Area-Unit [AREA-LABEL]
  |
  +-- Range-Value
        |
        +-- Density-Range
              HAS:
                Min-Value [Density]
                Max-Value [Density]
```

[UH/ISE Home Page | Phase One Report Index | Domain Model Example - Main Page]

Our Initial Architecture

Jump To: [Previous Page | Next Page]



[UH/ISE Home Page | Phase One Report Index]

Task Execution Interface

Jump To: [Previous Page | Next Page]

The task execution interface supports task selection, task specification, and task result retrieval. A task is a request to simulate a particular domain action.

- *Task selection* involves choosing a particular type of task to execute. It is accomplished in one of two ways:
  - The user browses an event hierarchy to locate the desired task type. The executable tasks are the leaf nodes of the hierarchy.
  - The user retrieves a existing task (using a constraint-based query mechanism). The input objects used for that task become the default input objects for the new instance of that task (the assumption is that the user will change some of these values).
- *Task specification* involves specifying values for the parameters of a selected task (instances of domain objects). It is accomplished in one of three ways.
  - By creating a new domain object of the needed type from scratch.
  - By directly using an existing object retrieved from existing instances of that object type.
  - By modifying an existing object retrieved from existing instances of that object type.
- *Task result retrieval* involves retrieving the results of previously executed tasks. The user does so by executing queries that specify constraints on previously executed tasks, the input parameters, and their output results.

[UH/ISE Home Page | Phase One Report Index]

An Example Of Task Execution

Jump To: [Previous Page | Next Page]

Given the domain model for *IMPACT*, the following actions are required to simulate an explosion:

- The user *selects* a particular type of `Breakup-Event` as the task (e.g., `Detonation`).
- The user *specifies* the values of the parameters the `Detonation` requires. This is the information necessary to construct the target and projectile objects (e.g., mass, position, etc...), as well as any detonation-specific parameters (e.g., energy available for detonation).
- These parameters are *mapped* into the input file *IMPACT* requires, with the various required place-holders and flags automatically placed into the file.
- The *IMPACT* program is *executed*, creating an output file.
- The output file is *mapped* into objects representing debris clouds (as well as fragment bins, masses, and any other produced objects).
- The user can then *retrieve* the contents of these debris clouds or the output from any other previously executed task.

Below are links to the specific objects the user must supply and can then access after executing the simulation.

- Executing An IMPACT Detonation: An Overview Of Its Input
- Executing An IMPACT Detonation: A Detailed Look At Its Input

[UH/ISE Home Page | Phase One Report Index]

Executing An IMPACT Detontation: An Overview Of Its Input

Jump To: [Next Page]

Assume the user has selected `Detonation` as the simulation task.

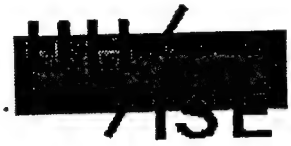
The user must then provide the the necessary input objects for this `Detonation` task. In particular, these are the high-level objects the user must supply (field name, type, and what leads to the need for the field):

`Epoch`	`Absolute-Time <- Breakup-Event`
`Min-Fragment`	`Length <- Breakup-Event`
`Target`	`Detonatable-Tank-Containing-Vehicle <- Explosion/Detonation`
`Available-Energy`	`Mass <- Detonation`
`%-Energy-To-Heat`	`Unitless-Value <- Explosion`
`%-Energy-To-Spreading`	`Unitless-Value <- Explosion`

These parameters can be supplied in any order and can be:

- Already-created objects.
- Newly-created objects.
- Variants of existing objects.

[UH/ISE Home Page | Phase One Report Index | Task Execution Example - Main Page]



The Domain-Object Repository

Jump To: [Previous Page | Next Page]

The *Domain Object Repository* (DOR) contains all user-created objects and user-executed tasks. Conceptually, it is essentially an object-oriented database.

All objects in the DOR have additional attributes that describes how and when the objects were created and in which tasks they were used.

- Creator (user who created object or executable task that produced it as output).
- Creation-Time (time/date when object was created).
- Tasks-Using (the tasks that take this object as input).

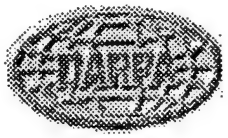
All tasks in the DOR have additional domain-independent attributes that describe who executed the task and when.

- Executor (user who executed task).
- Execution-Time (time/date when task was executed).

Finally, everything in the DOR (object or task) has several additional attributes:

- Name (a string identifier to support retrieval by name).
- Parent (an object or task that it was created as a variant of, if any).

[UH/ISE Home Page | Phase One Report Index]



The Basic Domain-Object Hierarchy

Jump To: [Previous Page | Next Page]

```
Entry
 |  HAS:
 |    Name [Unitless-Value]
 |    Parent [Entry]
 |
 +--Domain-Object
 |       HAS:
 |         Creator [Unitless-Value]
 |         Creation-Time [Absolute-Time]
 |         Tasks-Using [Collection of Domain-Action]
 |
 +--Domain-Action
         HAS:
           Executor [Unitless-Value]
           Execution-Time [Absolute-Time]
```

[UH/ISE Home Page | Phase One Report Index]

The Purpose Of The Domain-Object Repository

Jump To: [Previous Page | Next Page]

The Domain-Object Repository exists to support object and task retrieval so that:

- Retrieved objects can be fed into new tasks.

  Example: Using a satellite created for a glancing-blow collision in a catastrophic collision.

- Retrieved tasks can have their inputs slightly modified before being reexecuted.

  Example: Using a satellite created for a glancing-blow collision and modifying some of its attributes for another glancing-blow collision.

- Retrieved objects and tasks with similar properties can be examined for their similarities and differences.

  Example: Retrieving all satellites with a mass larger than 20kg to examine their other attributes to see what else was varied.

[UH/ISE Home Page | Phase One Report Index]

Support For Domain-Object Retrieval

Jump To: [Previous Page | Next Page]

At a minimum, the DOR must support retrieving objects and tasks by specifying:

- Entry type constraints

  Example: Retrieve the explosions.

- Combinations of entry type constraints with attribute value constraints (specific values, sets of values, and ranges of values)

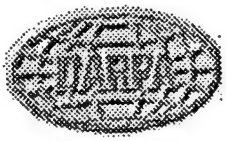
  Example: Retrieve the break-up events with target mass equal to 20kg.

  Example: Retrieve the collisions named Test-Col1, TestCol-4, and TestCol-6

  Example: Retrieve the explosions with a target mass greater than 20kg.

Retrieval essentially involves running through each instance of a specified type and verifying that its attributes meet the specified constriants.

[UH/ISE Home Page | Phase One Report Index]



Extending Retrieval To Intelligently Handle Units

Jump To: [Previous Page | Next Page]

Many objects contain two attributes: a value and a unit to use to interpret that value (e.g., the target mass is of type `Mass`, which is a numeric value and a unit of mass).

Retrieving all objects with a unit-based attribute is problematic since each instance may have a different unit used as part of a particular attribute's value

> *Example:* All `Boosters` with a mass greater than 20kg, where one booster can have a mass of 40lb and another of 5000g.

We address this *units problem* by:

- Extending the domain model to explicitly define conversions between instances of a unit type

  > *Example:* a "mass-object" is a unit type, and there are conversions defined between masses with different unit-values.

- Extending the retrieval process to check for and apply an optional conversion defined for a given attribute before checking the constraints involving that attribute.

  > *Example:* An attribute that's a mass will have its value converted to grams and then have the constraint applied to it.

[UH/ISE Home Page | Phase One Report Index]

Input/Output Mappings

Jump To: [Previous Page | Next Page]

Every task is associated with:

- An executable program or script to carry out the simulation.
- An "input mapping" that constructs the actual input files from the user-supplied domain objects. This involves:
  - Placing each attribute or parameter value in the appropriate location in the program's input.
  - Converting the attribute's value to the units expected by the program.
  - Placing any other values in the input files for the specified task.
- An "output mapping" that creates and fills in domain objects from the actual output files produced by the executed simulation. This involves:
  - Creating objects to hold the resulting output.
  - Locating the values of object attributes in the output files.

[UH/ISE Home Page | Phase One Report Index]

Support For Input/Output Mappings

Jump To: [Previous Page | Next Page]

In general, the mappings between object attributes and input and output files can be arbitrarily complex.

- The mapping process may require writing a complete program to transform a collection of domain objects into a simulation program's input files or to transform its output files into a collection of domain objects.
- In many cases, however, the mappings are much simpler, and can be handled by a relatively straightforward special-purpose language.

As a result, our approach is to support both ends of the spectrum by providing:

- A general API that a mapping program can use to access the database of domain-objects.
- Special-purpose languages for describing input and output mappings.

[UH/ISE Home Page | Phase One Report Index]

An API To Support Mapping

Jump To: [Previous Page | Next Page]

The API includes operations to:

- Access general information:
  - Retrieve the specific domain action being executed.
  - Retrieve the list of attributes for any object.
  - Retrieve the type of each attribute.
- Support input mappings.
  - Retrieve the value of a particular task or object attribute.
  - Retrieve the value of a paritcular field after converting it to a particular type of unit.
- Support output mappings.
  - Construct an object of a particular type,
  - Specify the value of an object's attribute.
  - Create a collection.
  - Add or delete objects from a collection,

[UH/ISE Home Page | Phase One Report Index]

Mapping Languages

Jump To: [Previous Page | Next Page]

There are two mapping languages:

- *An input mapping language* that describes how to place attribute values and constants in the input and that includes mechanisms for conditional placement of values and for handling collections of values.
- *An output mapping language* that describes how to take output values and place them in the input and that includes mechanisms for ignoring output values.

Below are links to an initial pass at these special purpose mapping languages and examples using them.

- A Language For Describing Input Mappings
- An Example Input Mapping: Explosions Into IMPACT
- An Actual IMPACT Input: Simulating An Explosion
- A Language For Describing Output Mappings
- An Example Output Mapping

[UH/ISE Home Page | Phase One Report Index]

A Language For Describing Input Mappings

Jump To: [Next Page]

The current language for describing input mappings is a small set of commands that focus on placing particular object attributes into the appropriate location in the appropriate input file.

- `value` places a *constant* in a particular place in the input file.

  `value` *file-name line-no line-pos specific-value*

- `item` exists to place an *attribute value* in a particular place in the input file (converting it to the appropriate unit if specified).

  `item` *file-name line-no line-pos attribute-path [unit-desc]*

- `items` exists to allow a collection of objects to be placed in the input file.

  `items` *file-name attribute-path*
  commands
  `end-items`

- `condition` exists to allow conditional placement and substitution of attributes and values in the input file.

  `condition` *attribute-path*
  `when` *value*
  commands
  `end-when`
  ...
  `end-condition`

The various terms are:

- *file-name* is an absolute file name (or a * to indicate the same file name as in the previous command).

- *line-no* is an absolute line number (a value) or a relative line number (a value preceded by a plus sign).

- *line-pos* is similar to *line-no*.
- *specific-value* is a constant to be placed in the specified position as is.
- *attribute-path* is of the form *field:field...:field* (e.g., Epoch:Absolute-Day:Week).
- *unit-desc* is one or more pairs of the form *attribute-name=unit-name* (e.g., Mass-Unit=kg).
- *commands* is one or more of the above commands.

[UH/ISE Home Page | Phase One Report Index | Mapping Languages - Main Page]

An Example Input Mapping: Explosions Into IMPACT

Jump To: [Previous Page | Next Page]

```
#
# EPOCH
#
value impact.in   0  0  "Epoch"
item  * +1  0  Epoch:Date:Year
item  * +0 +1  Epoch:Date:Month
item  * +0 +1  Epoch:Date:Day
item  * +0 +1  Epoch:Time-Within-Day:Hours:Actual-Value
item  * +0 +1  Epoch:Time-Within-Day:Minutes:Actual-Value
item  * +0 +1  Epoch:Time-Within-Day:Seconds:Actual-Value

#
# RANDOM SEED FOR FRAGMENTS/VELOCITIES
#
value * +1  0  "Random Number Seed"
value * +1  0  "3.42"

#
# TYPE OF BREAKUP-EVENT
#
value * +1  0  "1-Explosion, 2-Collision"
value * +1  0  "1"

#
# POSITION OF TARGET
#
condition Target:Orbital-Position:Reference
  when earth-centered
     value * +1 0 "ECI-Breakup Position (m)"
  end-when
  when spherical
     value * +1 0 "Spherical-Breakup Position (m)"
  end-when
  when classical-orbital
     value * +1 0 "Classical-Orbital Position (m)"
  end-when
end-condition
item * +1  0  Target:Orbital-Position:Position:X:Actual-Value Length-Unit=miles
item * +0 +1  Target:Orbital-Position:Position:Y:Actual-Value Length-Unit=miles
item * +0 +1  Target:Orbital-Position:Position:Z:Actual-Value Length-Unit=miles

#
# TYPE OF TARGET
#
value * +1 0 "Target type, 1-Booster, 2-PBV, 3-Satellite"
condition Target:Vehicle-Type
  when booster
    value * +1 0 "1"
  end-when
  when pbv
    value * +1 0 "2"
  end-when
  when satellite
    value * +1 0 "3"
  end-when
end-condition
```

```
#
# MASS OF TARGET
#
value * +1 0  "Target Mass (kg)"
item  * +1 0  Target:Mass:Actual-Value Mass-Unit=kg

#
# VELOCITY OF TARGET
#
condition Target:Orbital-Position:Velocity
  when earth-centered
     value * +1 0  "ECI-Pre-Breakup Velocity (m/s)"
  end-when
  when spherical
     value * +1 0  "Spherical Pre-Breakup Velocity (m/s)"
  end-when
  when classical-orbital
     value * +1 0  "Classical Pre-Breakup Velocity (m/s)"
  end-when
end-condition
item  *  +1  0  Target:Orbital-Position:Velocity:X:Actual-Value Length-Unit=miles
item  *  +0 +1  Target:Orbital-Position:Velocity:Y:Actual-Value Length-Unit=miles
item  *  +0 +1  Target:Orbital-Position:Velocity:Z:Actual-Value Length-Unit=miles

#
# LOSS OF ENERGY
#
value *  +1  0  "Fraction of energy to heat/spreading
item  *  +1  0  Target:%-Energy-To-Heat
item  *  +0 +1  Target:%-Energy-To-Spreading

#
# ADDITIONAL ENERGY (detonation ignores)
#
value *  +1  0  "Energy added to target/projectile"
value *  +1  0  "0 0"

#
# TYPE OF EXPLOSION
#
value *  +1  0  "Type of explosion 1-det, 2-pi"
value *  +1  0  "1"

#
# ENERGY FOR DETONATION
#
value *  +1  0  "Detonation Energy"
item  *  +1  0  Target:Available-Energy:Actual-Value Energy-Unit=joules

#
# TANK DESCRIPTION
#
value *  +1  0  "Dry Weight of Tank"
item  *  +1  0  Target:Tank:Dry-Weight:Actual-Value Mass-Unit=kg
value *  +1  0  "Material-Density of Tank"
item  *  +1  0  Target:Tank:Material-Density:Actual-Value Mass-Unit=kg Volume-Unit=cubic-meter
value *  +1  0  "Thickness Tank"
item  *  +1  0  Target:Tank:Thickness:Actual-Value Length-Unit=meters

#
# MISCELLANEOUS PARAMETERS TO AID OUTPUT GENERATION
#
value impact.in +1 0 "Index of minimum mass"
value impact.in +1 0 28
value impact.in +1 0 "Produce individual velocities"
value impact.in +1 0 "1"
value impact.in +1 0 "Produce individual rotations"
value impact.in +1 0 "1"
```

[UH/ISE Home Page | Phase One Report Index | Mapping Languages - Main Page]

An Actual IMPACT Input: Simulating An Explosion

Jump To: [Previous Page | Next Page]

```
Epoch
1991 2 20 10 45 0
Random Number Seed
3.42
1-Explosion, 2-Collision
1
ECI Breakup Position (m)
100 400 200
Target type, 1-Booster, 2-PBV, 3-Satellite
3
Target mass (kg)
100
ECI pre-breakup velocity (m/s)
100 400 200
Fraction of energy to heat/spreading
0.25 0.05
Energy added to target/projectile
0 0
Type of explosion 1-det, 2-pi
1
Detonation Energy
1000
Dry Weight of Tank (kg)
20
Material Density of Tank
.50
Thickness of Tank (cm)
2.0
Index of minimum mass
28
Produce individual velocities
1
Produce individual rotations
1
```

[UH/ISE Home Page | Phase One Report Index | Mappings Languages - Main Page]

A Language For Describing Output Mappings

Jump To: [Previous Page | Next Page]

The current language for describing output mappings is a small set of commands.

- `skip` ignores input values (by ignoring a specified number of lines and then a specified number of values).

  `skip` *file-name lines-to-ignore values-to-ignore*

- `place` takes the next input value and puts it in the specified attribute.

  `place` *file-name attribute-path*

- `set` places a constant in a specified attribute.

  `set` *attribute-path constant-value*

- `repeat` executes a series of commands a fixed number of times, making available the repetition count.

  `repeat` *count-variable start-count end-count*
  commands
  `end-repeat`

- `create` exists to allow the creation of an object of a particular type.

  `create` *object-name object-type*

- `append` adds an object to the collection of objects making up an attribute's value.

  `append` *object-name attribute-path*

The various terms are:

- *file-name* is an absolute file name.
- *lines-to-ignore* is the number of lines to ignore (it can be zero).

- *values-to-ignore* is the number of additional values to be ignored after the specified number of lines have been ignored (and it can also be zero).
- *count-variable* is a variable that holds the current repeat count (which is updated by 1 each the commands are repeated). The variable only has meaning within the repeat construct.
- *start-count* is the initial value of *count-variable*.
- *end-count* is the final allowed value of *count-variable*.
- *object-name* is a way to refer to a newly created object.
- *object-type* is the name of a type in the domain model.
- *attribute-path* can be in one of two forms. It can be the form *field:field...:field* (e.g., Epoch:Absolute-Day:Week), which defaults to starting with a field in the currently defined task. Or it can be of the form *name!field:field...:field* (e.g., New-DC:Num-Fragments).
- *commands* is one or more of the above commands.

[UH/ISE Home Page | Phase One Report Index | Mapping Languages - Main Page]

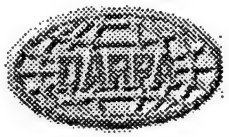
An Example Output Mapping: IMPACT Into Collision

Jump To: [Previous Page]

```
#
# Skip initial junky output (7 lines, plus 2 values)
#
skip impact.out 7 2

#
# Save Mass-Transfer as a result of collision, as well
# as Relative-Velocity of target after collision
#
place * Mass-Transfer:Actual-Value
set Mass-Transfer:Mass-Unit kg
place * Relative-Velocity
set Relative-Velocity:Distance-Unit meters
set Relative-Velocity:Time-Unit seconds

#
# Save Mass-Bin by creating each individual Mass-Bin-Entry,
# and then adding it to the collection of Mass-Bin-Entries.
#
skip * 11 0
place * Fragment-Mass-Bin:Location-Of-Smallest
skip * 1
repeat Next-Bin-Number 1 Fragment-Mass-Bin:Location-Of-Smallest
  create New-Bin-Entry Fragment-Bin

  set New-Bin-Entry!Entry-Index Next-Bin

  place * New-Bin-Entry!Max-Fragment-Mass:Actual-Value
  set New-Bin-Entry!Max-Fragment-Mass:Mass-Unit kg

  place * New-Bin-Entry!Max-Fragment-Size:Actual-Value
  set New-Bin-Entry!Max-Fragment-Size:Length-Unit cm

  place * New-Bin-Entry!Min-Fragment-Size:Actual-Value
  set New-Bin-Entry!Min-Fragment-Size:Length-Unit cm

  append New-Bin-Entry Fragment-Mass-Bin:Entries
end-repeat

#
# Save number of debris clouds
#
skip * 5 0
place * Number-Of-Debris-Clouds

repeat Next-Cloud-Number 1 Number-Of-Debris-Clouds
#
# Save all of the info for each debris cloud
#
  create New-DC Debris-Cloud

  place * New-DC!Post-Collision-Velocity:Actual-Value
  set New-DC!Post-Collision-Velocity:Distance-Unit meters
  set New-DC!Post-Collision-Velocity:Time-Unit sections

  skip * 2 0
```

```
place * New-DC!KE-To-Heat
place * New-DC!KE-To-Spread

skip * 7 0
place * New-DC!Bin-With-Largest
set New-DC!Bin-With-Smallest Fragment-Mass-Bins:Num-Bins

#
# Save the fragment groups for a debris cloud
#
repeat Next-Bin-Num New-DC:Bin-With-Largest New-DC:Bin-With-Smallest
  create New-Fragment-Group Fragment-SV-Group

  skip * 3 0
  place * New-Fragment-Group!Bin-Containing-Group
  place * New-Fragment-Group!Num-Fragments
  place * New-Fragment-Group!Characteristic-Velocity:Actual-Value
  set New-Fragment-Group!Characteristic-Velocity:Distance-Unit meters
  set New-Fragment-Group!Characteristic-Velocity:Time-Unit seconds
  skip * 1 0
  place * New-Fragment-Group!Num-Spread-Velocities

  #
  # Save the individual spread velocities
  #
  skip * 1 0
  repeat Next-Spread-Vel-Num 1 New-Fragment-Group!Num-Spread-Velocities
    create New-Spread-Velocity-Group Spread-Velocity-Group
    place * New-Spread-Velocity-Group!Spread-Velocity
    place * New-Spread-Velocity-Group!Num-Fragments
    append New-Spread-Velocity-Group New-Fragment-Group!Entries
  end-repeat

  append New-Fragment-Group New-DC!Fragment-Spread-Velocities

  #
  # Save the individual fragment mass and full velocities
  #
  skip * 1 0
  place * New-DC!Num-Fragments
  skip * 0 1
  place * New-DC!Min-Fragment-Size:Actual-Value
  set New-DC!Min-Fragment-Size:Length-Unit meters
  skip * 0 3
  skip * 1 0
  repeat Next-Fragment-Num 1 New-DC!Num-Fragments
    create New-Fragment
    place * New-Fragment:Mass-Bin
    place * New-Fragment:Velocity:Xv:Actual-Value
    set New-Fragment:Velocity:Xv:Time-Unit meters
    set New-Fragment:Velocity:Xv:Distance-Unit second
    place * New-Fragment:Velocity:Yv:Actual-Value
    set New-Fragment:Velocity:Yv:Time-Unit meters
    set New-Fragment:Velocity:Yv:Distance-Unit second
    place * New-Fragment:Velocity:Zv:Actual-Value
    set New-Fragment:Velocity:Zv:Time-Unit meters
    set New-Fragment:Velocity:Zv:Distance-Unit second
    append New-Fragment New-DC!Fragments
  end-repeat
 end-repeat
 append New-DC Debris-Clouds
end-repeat
```

[UH/ISE Home Page | Phase One Report Index | Mapping Languages - Main Page]



How Simulation Input and Output Is Processed

Jump To: [Previous Page | Next Page]

Simulation input is generated by invoking a mapping program to generate the necessary input files. The mapping program:

- Can be either a regular executable that uses the API or a program written using the special-purpose mapping language.
- Must take care of unit conversions, either implicitly using constructs in the mapping language or explicitly within programs using the API.

Simulation output is processed in two steps.

- All of the individual domain objects needed to hold the program's resulting output are automatically created, except for those that are organized in collections.
- The mapping program is then run to fill in their attributes, and to create and fill in objects that appear in collections.

Processing simulation object appears to be more complex than producing simulation input.

- Simulation input involves taking attribute values and placing them in input files.
- Simulation output involves creating objects and collection of objects,

The result is that our simple language to support output mappings is more complex than that for input processing.

[UH/ISE Home Page | Phase One Report Index]

Making Our Initial Approach Work

Jump To: [Previous Page | Next Page]

Given a set of simulation programs, it is necessary to perform the following actions to support their use:

- Enumerate the various tasks that these simulation programs are used to simulate. These tasks correspond to the domain actions simulated by these programs.

  We initially assume a one-to-many mapping between simulation programs and tasks (i.e., a single program or script can perform many tasks, depending on options selected or particular input values given). We do not address the case where there are many simulation programs that perform the same general action.

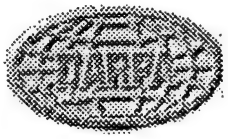
- Determine the parameters required by each task and the output values each task produces. The types of individual and groups of parameters and output values correspond to the domain objects.

  We fundamentally assume that there are usually high-level groupings of both parameters and output values that help organize specific lower-level input and output values.

- Provide input and output mappings from object/attributes to and from the specific values required by the executing program.

  We fundamentally assume that in practice there are straightforward mappings between the domain model and the actual input and output, even though in general the input and output mappings could be arbitrarily complex.

[UH/ISE Home Page | Phase One Report Index]



Costs Of Our Initial Approach

Jump To: [Previous Page | Next Page]

There are two key costs to this approach.

- Time and effort to construct the domain model. This involves determining which tasks each program performs and grouping its input and output parameters into higher-level abstractions.

  For IMPACT, this took approximately 2 person-days to cover the set of key tasks the program performs.

- Time and effort to construct the input and output mappings. This involves determining exactly what type of input/output formats the simulation program requires and and generates, and it involves writing the mapping programs to produce them from the domain model.

  For IMPACT, this took approximately 3 person-days to cover the set of key tasks the program performs.

In general, both of these tasks may require examining program documentation, sample input and output, and possibly the source code.

These costs are reduced by a pair of factors.

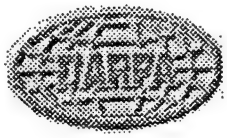
- There is potential of reuse for the domain model, as the domain model is not constructed from scratch each time.

  With IMPACT's current domain model, approximately 20% of it can be reused across most simulation programs (time, units, and so on) and another 30% can be reused across programs in the same domain (debris clouds, fragments, and so on).

- There is potential reuse of the input/output mappings. Once the mappings have been done for one of the tasks a program performs, the other mappings are much quicker.

  With IMPACT, over 80% of the input mapping description needed for a detonation can be reused for a pressure-induced explosion.

[UH/ISE Home Page | Phase One Report Index]



Analysis Of Our Initial Approach

Jump To: [Previous Page]

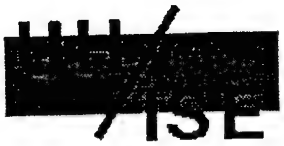
Our initial approach, applied only to simulation programs in isolation, provides several important benefits:

- Users are isolated from the ugly details of running programs.
- Users can reuse existing input data at arbitrary levels of generality.
- Users no longer need to explicitly manage simulation results.
- Users can perform content-based retrieval of previously executed simulation results.

However, our initial environment has several drawbacks that must be addressed before deploying it to tackle simulation programs in isolation.

- The task selection mechanism is too simplistic.
- The input/output mapping languages are too tied to our initial simulation programs.
- The domain-object retrieval query mechanism is too weak.

[UH/ISE Home Page | Phase One Report Index]



Benefit: Isolation From Simulation Details

Jump To: [Next Page]

Our approach hides from the user many of the details of actually setting up and running simulation programs and then interpreting their results.

- Users are able to select tasks based solely on function.
- Users no longer have to worry about file formats, special input parameters, unit values, and so on.

An obvious alternative (that doesn't require a domain model) is to provide a GUI-based script that requests all of the values necessary to execute the program and sets up the program's input files.

Our approach (in the simplest view) can be thought of as a mechanism to extend this alternative by:

- *Automatically* generating the necessary GUI from a high-level description of the program's input (the domain model).
- *Semi-automatically* transforming the input to what the program requires (the input mapping).

However, this support for a nicer interface to an existing simulation programs is only one of the features of having an explicit domain model.

[UH/ISE Home Page | Phase One Report Index | Analysis - Main Page]

Benefit: Ability To Reuse Simulation Input And Output

Jump To: [Previous Page | Next Page]

Our approach allows users to reuse existing simulation program inputs and outputs at different levels of granularity.

- Users can create an object once (as the input to one execution of a simulation task) and use it as input to multiple simulation tasks.
- Users can modify existing objects and use them as inputs to new simulation tasks.

There are several obvious alternatives.

- Let users simply reuse entire input/output files by copying and editing.

  However, this forces users to worry about low-level file formats.

- Have a GUI provide previously used values for each input parameter and allow the user to select the desired value.

  However, this breaks down when programs have many parameters or when the individual parameters have many past values. One way to avoid this breakdown is to group parameters, which is exactly what our object-oriented approach does.

In essence, this reuse supports simplifying the execution of a variety of different simulations, each differing in only a portion of their parameters.

[UH/ISE Home Page | Phase One Report Index | Analysis - Main Page]

Benefit: Simplified Management Of Simulation Results

Jump To: [Previous Page | Next Page]

Our approach greatly simplifies the management of simulation results:

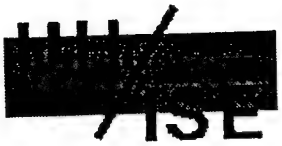
- By automatically gathering results into objects and attributes.
- By supporting retrieval of results based on constraints on object and attribute values.
- By automatically maintaining relationships between different simulation runs.

One alternative approach is to have scripts automatically add web page links between a simulation program, its input files, and its output results. However:

- This requires user assistance to describe links to facilitate later retrieval (e.g., naming a particular run with its purpose).
- There is no structure between different runs (e.g., that a particular set of runs involved varying several particular high-level parameters).
- It doesn't facilitate breaking up the input/output into smaller pieces (e.g., if only one or two of the output values are the ones of interest).

Essentially, our approach eliminates the need for explicit user management of results.

[UH/ISE Home Page | Phase One Report Index | Analysis - Main Page]



Benefit: Content-Based Retrieval

Jump To: [Previous Page | Next Page]

Our approach supports the retrieval of simulation tasks, inputs, and results based on content. In particular, users specify properties of task input or output and the system retrieves those entities that meet those properties.

There are two alternatives:

- Have the user label simulation inputs, output, and tasks and retrieve based on those labels.

  The idea here is that the labels somehow capture the key aspects of content that would be used for later retrieval. This is difficult to do in advance.

- Have the user use lower-level tools (e.g., "grep") on the raw simulation input and output.

  Unfortunately, it is also often very difficult to specify just the particular items of interest using these tool, as well as to retrieve related data not explicitly matched within the tool.

Essentially, our approach allows for specific retrievals without the need for labels or the use of matching tools.

[UH/ISE Home Page | Phase One Report Index | Analysis - Main Page]

Shortcoming: Problems With Task Selection

Jump To: [Previous Page | Next Page]

In our approach, task selection is done by the user using a standard "tree" browser to examine the event hierarchy. The user then selects a task by selecting a leaf node of the tree.

There are two problems with this approach.

- It forces the user to select a very specific task.

  This is problematic because the user may not be able to distinguish which leaf node is desired, because which node should be selected may depend on particular relationships between input parameters.

  For example, the user may only desire a collision, without knowing whether it should be catastrophic, a glancing blow, and so on. That's because the particular type of collision depends on the relative kinetic energies of the target and projectile.

- It forces the user to know the name of the task to select.

  This is problematic because the user may want to select a task by functionality.

  For example, the user may want to produce a debris cloud and wants to select a particular task from a list of all tasks that are capable of doing so.

[UH/ISE Home Page | Phase One Report Index | Analysis - Main Page]

Shortcoming: Problems With I/O Mapping Languages

Jump To: [Previous Page | Next Page]

Our mapping languages were not designed to be general. Instead, they were developed in an ad-hoc manner to be just powerful enough to support automating the mappings for the particular simulation programs we examined.

This leads to a variety of I/O mappings that our I/O mapping languages can't handle, but that would be required to apply our model to other real-world simulation programs.

The most important of these language shortcomings appear to be:

- Our input mappings allows us to take an entire collection and dump it to an input file. It does not allow us to deal with only selected portions of a collection.
- Our output mappings do not allow us to conditionally place the output in different attributes or different objects.

Essentially, these problems point out that it is worth some effort to explore how to develop more general input and output mapping languages.

[UH/ISE Home Page | Phase One Report Index | Analysis - Main Page]

Shortcoming: Weaknesses Of Our Domain-Object Retrieval Mechanism

Jump To: [Previous Page]

It is clearly necessary to provide some mechanism for retrieving objects and tasks by task type and attribute value. And the more powerful the queries supported by the retrieval mechanism, the easier it is for the user to retrieve precisely the desired objects.

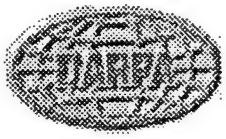
Our initial retrieval mechanism is primitive, except for its sophisticated handling of "units". It really serves only to illustrate the usefulness of object-retrieval within the context of our larger simulation environment.

It falls short of being a general retrieval mechanism in two obvious places:

- Not supporting retrievals that involve contains on the relationships between attributes (e.g., "retrieve all collisions with a target whose mass is less than that of its projectile").
- Not supporting logical relationships other than "AND" between attributes (e.g. "retrieve all collisions with a target whose mass is greater than 100kg or whose projectile's mass is less than 50kg").

Handling the logical relationships is a straightforward extension. Handling the relationship-based constraints is less straightforward, as this extension can't be implemented by having the user simply fill in ranges on individual object attibutes.

[UH/ISE Home Page | Phase One Report Index | Analysis - Main Page]



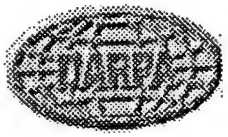
Project Report: Phase Two

Extensions To An Architecture Using a Domain Model To Support Simulation Execution and Simulation Data and Results Management

---

- The Focus Of These Extensions
- Overview Of Task Sequences
- Purpose Of Task Sequences
- System-Maintained Task Sequences
- Example System-Maintained Task Sequences
- User-Constructed Task Sequences
- Example User-Constructed Task Sequences
- Task Sequence Browsing
- Examples Of Task Sequence Browsing
- Benefits Of Sequence Browsing
- Sequence Overviews
- Examples Of Task Sequence Overviews
- Alternatives To Task Sequences
- Integrating Simulation Programs
- File-Level Integration
- Domain-Level Integration
- Supporting Domain-Level Integration With Views
- Supporting Simulation Program Evolution
- Additional Integration Support

[UH/ISE Home Page]



Purpose Of Task Sequences

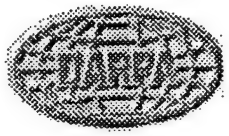
Jump To: [Previous Page | Next Page]

Task sequences represent a subset of a history of simulation actions.

They support asking and answering high-level questions about that history.

- What specific simulation actions were done and what parameters did they use?
- What actions and paramters led to a particular conclusion?
- What actions weren't done or what parameters weren't tried?
- What was common and different among the actions that were done?

[UH/ISE Home Page | Phase Two Report Index]



Overview Of Task Sequences

Jump To: [Next Page]

The Domain Object Repository is an unorganized collection of domain objects and actions.

A useful extension is to add an organizational mechanism called *task sequences* on top of the domain actions (tasks) in the repository.

A task sequence is a named collection of ordered actions. There are two types of task sequences:

- Temporally-ordered: organized so the domain actions simulated first appear before those simulated later.
- Attribute-sorted: organized according to values of particular attributes of the domain actions.

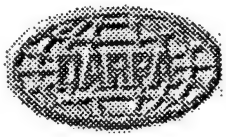
Some example task sequences are:

> *The collection of all explosions with a target mass greater than 10kg, organized temporally from first to last (in the order executed).*
>
> *The collection of all glancing blow collisions, sorted by projectile mass.*

All task sequences are stored in the *Task Sequence Repository* (TSR), where they can be retrieved by name or by browsing.

Task sequences are either user system-maintained or user-constructed.

[UH/ISE Home Page | Phase Two Report Index]



System-Maintained Task Sequences

Jump To: [Previous Page | Next Page]

System-maintained task sequences are automatically updated after each executed domain action.

They effectively form a history of the user's actions.

They include:

- *A sequence of all user-executed actions, in temporal order.*

  This sequence is labelled the *Task-History-Sequence*, and it forms a *conceptual-level* audit trail of all simulation actions done by a user.

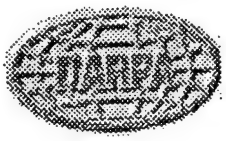
  One use is to form a report on exactly what the user did to reach a particular conclusion.

- *One sequence per action type of all user-executed actions of that type, again in temporal order.*

  These sequences are labeled "*type*-history-sequence" and they provide action-specific audit trails (e.g., all explosions simulated by the user).

  One use is to make it easy to understand how exactly specific user actions differed.

[UH/ISE Home Page | Phase Two Report Index]



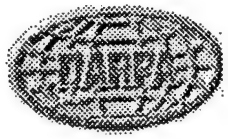
Example System-Maintained Task Sequences

Jump To: [Previous Page | Next Page]

Here are some examples of different types of system-maintained task sequences.

- A "task-history-sequence".
- Its corresponding type-history-sequence's:
  - The "Breakup-Event-History-Sequence" is identical to the "Task-History-Sequence" (assuming that only different types of breakup events have been executed).
  - The explosion-related history sequences.
  - The collision-related history sequences.

[UH/ISE Home Page | Phase Two Report Index]



An Example Task-History Sequence

Jump To: [Next Page]

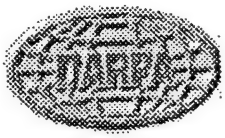
The "Task-History-Sequence" (viewed with a depth of zero, and using the default names for each entry).

```
Pressure-Induced-Explosion-1
Pressure-Induced-Explosion-2
Pressure-Induced-Explosion-3
Pressure-Induced-Explosion-4
Detonation-5
Detonation-6
Detonation-7
Booster-Glancing-Blow-Collision-8
Booster-Glancing-Blow-Collision-9
Non-Booster-Glancing-Blow-Collision-10
Pressure-Induced-Explosion-11
Detonation-12
Non-Booster-Glancing-Blow-Collision-13
Non-Booster-Glancing-Blow-Collision-14
Non-Booster-Glancing-Blow-Collision-15
```

Sequences can be viewed at any depth. Here is an entry in the above sequence viewed at a depth of one:

```
Pressure-Induced-Explosion-1
  INPUT
    Epoch [Absolute-Time-1]
    Min-Fragment [Length-1]
    Target [Detonatable-Tank-Containing-Vehicle-1]
    Available-Energy [Energy-1]
    Time-To-Act [Relative-Time-1]
    Pressure-Build-Up [Pressure-1]
  OUTPUT
    Fragments [Debris-Cloud-1]
```

[UH/ISE Home Page | Phase Two Report Index | Sequence Examples - Main Page]



Example Explosion History Sequences

Jump To: [Previous Page | Next Page]

The "Explosion-History-Sequence":

```
Pressure-Induced-Explosion-1
Pressure-Induced-Explosion-2
Pressure-Induced-Explosion-3
Pressure-Induced-Explosion-4
Detonation-5
Detonation-6
Detonation-7
Pressure-Induced-Explosion-11
Detonation-12
```

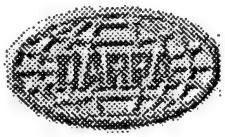
The "Detonation-Sequence":

```
Detonation-5
Detonation-6
Detonation-7
Detonation-12
```

The "Pressure-Induced-Explosion-History-Sequence":

```
Pressure-Induced-Explosion-1
Pressure-Induced-Explosion-2
Pressure-Induced-Explosion-3
Pressure-Induced-Explosion-4
Pressure-Induced-Explosion-11
```

[UH/ISE Home Page | Phase Two Report Index | Sequence Examples - Main Page]



Example Collision History Sequences

Jump To: [Previous Page]

The "Collision-History-Sequence".

```
Booster-Glancing-Blow-Collision-8
Booster-Glancing-Blow-Collision-9
Non-Booster-Glancing-Blow-Collision-10
Non-Booster-Glancing-Blow-Collision-13
Non-Booster-Glancing-Blow-Collision-14
Non-Booster-Glancing-Blow-Collision-15
```

The "Glancing-Blow-History-Sequence" is identical to the "Collision-History-Sequence", since all Collisions in our example are Glancing-Blow-Collsions.

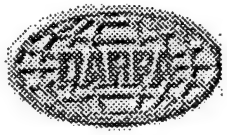
The "Booster-Glancing-Blow-History-Sequence":

```
Booster-Glancing-Blow-Collision-8
Booster-Glancing-Blow-Collision-9
```

The "Non-Booster-Glancing-Blow-History-Sequence":

```
Non-Booster-Glancing-Blow-Collision-10
Non-Booster-Glancing-Blow-Collision-13
Non-Booster-Glancing-Blow-Collision-14
Non-Booster-Glancing-Blow-Collision-15
```

[UH/ISE Home Page | Phase Two Report Index | Sequence Examples - Main Page]



User-Constructed Task Sequences

Jump To: [Previous Page | Next Page]

User-constructed task sequences have several purposes:

- They capture answers to queries about the actions being done.
- They allow users to group past actions according to particular properties.

There are four types of user-constructed sequences:

- *A sequence of actions that results from a query, in temporal order* (e.g., all collisions with a mass in a particular range and a projectile mass in another range).

  The user constructs it by specifying a set of constraints on the attributes of actions in an existing sequence.

- *A sequence of actions that are temporally related to a particular action* (e.g., the set of explosions preceding, either directly or indirectly, a particular explosion).

  The user constructs it by specifying a sequence, an action, and the desired temporal relationships (e.g., preceding, following, and so on).

- *A sequence of actions that results from a set operation, in temporal order* (e.g., all explosions that are not pressure-induced explosions with a target mass less than 10kg).

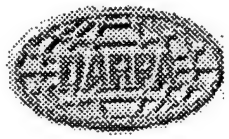
  The user constructs it by applying union, intersection, or difference to a pair of existing sequences.

- *A sequence of actions sorted according to user-supplied sorting criteria* (e.g., ordering a of explosions by target mass).

  The user constructs it by specifying an an existing sequence and a list of attributes to use as the sort key (the primary key, the secondary key, and so on).

After these sequences are constructed, they are assigned a name of the form "`user-`*Name*`-sequence`", where *Name* is provided by the user.

[UH/ISE Home Page | Phase Two Report Index]



Example User-Constructed Task Sequences

Jump To: [Previous Page | Next Page]

Query Constructed: All explosions with target mass less than 10kg.

```
Pressure-Induced-Explosion-3
Pressure-Induced-Explosion-4
Detonation-7
Pressure-Induced-Explosion-11
Detonation-12
```

Temporal Chain: All explosions directly connected to Pressure-Induced-Explosion-4.

```
Pressure-Induced-Explosion-1
Pressure-Induced-Explosion-2
Pressure-Induced-Explosion-3
Pressure-Induced-Explosion-4
Detonation-5
Detonation-6
Detonation-7
```

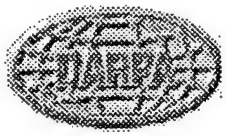
Set Constructed: All items in the first but not the second.

```
Pressure-Induced-Explosion-3
Pressure-Induced-Explosion-4
Detonation-7
```

Attribute Ordered: Sort the previous set by target mass (in ascending order).

```
Pressure-Induced-Explosion-3
Detonation-7
Pressure-Induced-Explosion-4
```

[UH/ISE Home Page | Phase Two Report Index]



Task Sequence Browsing

Jump To: [Previous Page | Next Page]

All task sequences support browsing (running through the actions they contain in a forward or backward direction).

Each action is displayed during browsing as a hierachical structure of fields and values. The user can control the depth of the hierarchy that's displayed.

Browsing a sequence automatically highlights the differences between successive items in a task sequence. There are three cases:

- *The successive items are the same type* (e.g., a sequence of Non-Catastrophic Collisions).

  The browser simply highlights the fields containing different values from the previous element (e.g., the target mass).

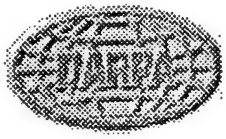
- *The successive items are different types, but share an ancestor* (e.g., a sequence of Explosions containing a combination of Pressure-Induced Explosions and Detonations).

  The browser divides each action into shared fields and unshared fields and highlights the name of the shared fields containing different values from the previous action.

- *The successive items share no ancestor* (e.g., the "Task-History-Sequence").

  The browser leaves the shared fields section empty and places all fields in the unshared section.

[UH/ISE Home Page | Phase Two Report Index]



Examples Of Task-Sequence Browsing

Jump To: [Previous Page | Next Page]

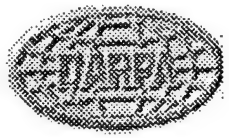
We provide a pair of browsing examples. In these examples, when moving from an action *X* to an action *Y*:

- Unshared attributes (those in *X* but not in *Y* are indicated with a plus sign).
- Shared attributes (those in both *X* and *Y*) with differing values are indicated with an asterisk.
- Shared attributes with the same values have nothing preceding them.

The examples show:

- Browsing from one instance of a type to another instance of the same type (from Pressure-Induced-Explosion-1 to Pressure-Induced-Explosion-2).
- Browsing from one instance of a type to an instance of a different type (from Detonation-5 to Pressure-Induced-Explosion-4).

[UH/ISE Home Page | Phase Two Report Index]



Browsing Shared Types

Jump To: [Next Page]

- The user initially views "Pressure-Induced-Explosion-1".

```
Pressure-Induced-Explosion-1
  INPUT
    Epoch [Absolute-Time-1]
    Min-Fragment [Length-1]
    Target [Detonatable-Tank-Containing-Vehicle-2]
    Available-Energy [Energy-1]
    Time-To-Act [Relative-Time-1]
    Pressure-Build-Up [Pressure-1]
  OUTPUT
    Fragments [Debris-Cloud-1]
```

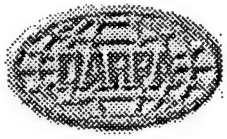
- The user then moves to view "Pressure-Induced-Explosion-2", with the two attributes that differ clearly highlighted.

```
Pressure-Induced-Explosion-2 [from: Pressure-Induced-Explosion-1]
  INPUT
    Epoch [Absolute-Time-1]
    Min-Fragment [Length-1]
  * Target [Detonatable-Tank-Containing-Vehicle-2]
    Available-Energy [Energy-1]
    Time-To-Act [Relative-Time-1]
    Pressure-Build-Up [Pressure-1]
  OUTPUT
  * Fragments [Debris-Cloud-2]
```

- This view shows at a *high-level* what was changed between the two items (and importantly, what remained the same).

- The user can then examine the particular Vehicles and Debris-Clouds to determine the precise differences.

[UH/ISE Home Page | Phase Two Report Index | Browsing Examples - Main Page]



Browsing Differing Types

Jump To: [Previous Page]

The user first views "Detonation-5".

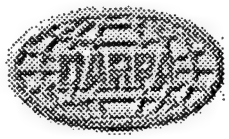
```
Detonation-5
  INPUT
    Epoch [Absolute-Time-8]
    Min-Fragment [Length-16]
    Target [Detonatable-Tank-Containing-Vehicle-3]
    Available-Energy [Energy-2]
    %-Energy-To-Heat [20]
    %-Energy-To-Spreading [80]
  OUTPUT
    Fragments [Debris-Cloud-5]
```

The user then views "Pressure-Induced-Explosion-4" after having viewed "Detonation-5".

```
Pressure-Induced-Explosion-4 [from: Detonation-5]
  INPUT
    Epoch [Absolute-Time-8]
    Min-Fragment [Length-14]
  * Target [Detonatable-Tank-Containing-Vehicle-2]
    Available-Energy [Energy-2]
  + Time-To-Act [Relative-Time-12]
  + Pressure-Build-Up [Pressure-4]
  OUTPUT
  * Fragments [Debris-Cloud-4]
```

The user sees that of the parts that were common between these simulations, the differences were in the Target and the resulting Debris-Could.

[UH/ISE Home Page | Phase Two Report Index | Browsing Examples - Main Page]



Benefits Of Task Sequence Browsing

Jump To: [Previous Page | Next Page]

All task sequences support browsing (the ability to run through the actions a sequence contains in a forward or backward direction).

The ability to browse sequences aids the user in:

- Determining what is changing between successive simulation actions (e.g., identifying the particular parameter(s) being studied in a parametric study).

  Example: Recognizing that the user repeatedly tried different target masses in a series of pressure-induced explosions (possibly to try to generate a debris cloud that matches an existing debris cloud).

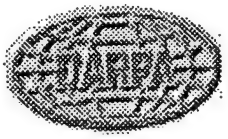
- More easily recognizing patterns of actions (e.g., noticing that certain sequences of actions are repeatedly executed).

  Example: Recognizing that the user repeatedly executing a pressure-induced explosion followed by a detonation for the same basic set of parameters (possibly to understand the differences in the effects of the two actions).

- Understanding the process by which a particular conclusion was reached (e.g., noticing how and when parameters were changed as part of a parametric study).

  Example: Recognizing that the user first varied the target mass of a pressure-induced explosion, then varied the pressure buildup (possibly to first get a course match, then doing fine tuning).

[UH/ISE Home Page | Phase Two Report Index]



Task Sequence Overviews

Jump To: [Previous Page | Next Page]

All task sequences support *overviews*. An overview is a summary of the actions the sequence contains.

Currently, there are two types of overviews.

- *A "commonality/difference" summary*: an overview of what's common and different between elements in the sequence.

  It summarizes what's shared in the sequence by type (e.g., a shared type or shared ancestor of all sequence actions, if any) and by attribute (e.g., the attributes that all actions have in common and that contain the same values).

  It also summarizes the differences by attribute (e.g., the attributes that all actions have in common but whose value differs among the actions).

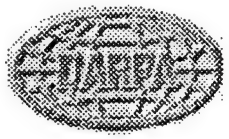
  This overview is useful for determining which parameter was varied in a particular sequence of actions.

- *A "size" summary*: an overview of the number of items of each type in the sequence, organized by type.

  This overview is useful for determining what actions a particular sequence emphasized.

There are many other possible overviews (such as organizing the seqeunce by attributes and attribute values), as well as variants on these overviews (such as providing attribute ranges for shared attributes with unshared attribute values).

[UH/ISE Home Page | Phase Two Report Index]



Example Task Sequence Overviews

Jump To: [Previous Page | Next Page]

Two possible task sequence overviews are:

- *Commonality/Difference Summary* of "Pressure-Induced-Explosion-History-Sequence".

```
SHARED AMONG SEQUENCE ELEMENTS:
  TYPE INFO:
    Breakup-Event
      Explosion
        Pressure-Induced-Explosion
  ATTRIBUTES:
    INPUT
      Epoch [Absolute-Time-1]
      Min-Fragment [Length-1]
      Available-Energy [Energy-1]
      Time-To-Act [Relative-Time-1]
      Pressure-Build-Up [Pressure-1]

DIFFERENT:
  ATTRIBUTES:
    INPUT
      Target
    OUTPUT
      Fragments
```

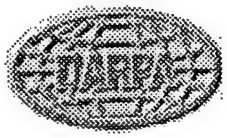
  This overview makes it clear that only properties of the target were changed.

- *Size Summary* of the "Task-History-Sequence".

```
Breakup-Event 15
  Explosion 9
    Pressure-Induced-Explosion 5
    Detonation 4
  Collision 6
    Glancing-Blow-Collision 6
      Non-Booster-Glancing-Blow-Collision 4
      Booster-Glancing-Blow-Collision 2
    Catastrophic-Collision 0
    Non-Catastrophic-Collision 0
    Head-On-Collision 0
```

  This overview makes it clear that explosions were equally distributed between the two types, while collisions were focused on Glancing-Blow-Collision variants.

[UH/ISE Home Page | Phase Two Report Index | Sequence Examples - Main Page]



Alternatives To Task Sequences

Jump To: [Previous Page | Next Page]

The obvious alternative to task sequences for capturing and accessing historical information is to:

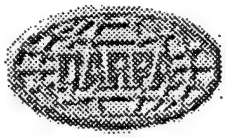
- Maintain a log of commands executed and files accessed, along with the contents of those files.
- Examine commands to determine what programs were executed.
- Perform file "diffs" to determine how input changed between program runs.

This alternative is problematic, as it focuses on low-level details (commands and files) and not higher-level actions (simulation tasks and their conceptual parameters).

It therefore cannot conveniently answer high-level questions about simulation executions:

- Determining which high-level tasks were simulated requires examining both the commands and their input files.
- Determining which parameters were varied requires examining files and file "diffs" and trying to map low-level values to higher-level attributes.

[UH/ISE Home Page | Phase Two Report Index]



Integrating Simulation Programs

Jump To: [Previous Page | Next Page]

We have so far focused on using a domain model to support the repeated execution of a single simulation program, concentrating on:

- Simplifying the use of the program.
- Allowing the reuse of input values across simulations.
- Providing a domain-level history of how that program was used.

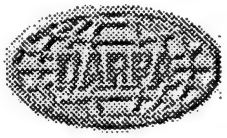
However, we often want simulation programs in the same domain to take advantage of each other's results.

The explicit domain model provides support for integrating multiple simulation programs and related tools.

We consider supporting integration for two different types of situations:

- *There already exist a set of simulation tools designed to work together.* Typically, these tools are already integrated at the file and program level.
- *There already exist a set of simulation tools in the same domain but not originally designed to work together.* Often, these tools can be integrated at the file and program level with the use of intermediate programs to transform existing results and to request additional data.

[UH/ISE Home Page | Phase Two Report Index]



File-Level Integration

Jump To: [Previous Page | Next Page]

One example of file/program level integration is:

> The *Impact* program produces a data file describing the particles that result from a simulated breakup event. This data file is then input (along with other information, including a catalog containing the orbital positions of various space-based objects) to a program called *Debris*, which computes the orbits of the particules and determines the probability of collision with existing space objects (based on catalogs of objects and positions).

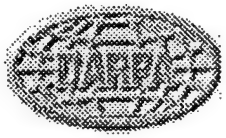
In general, with file-level integration, the process for executing a set of related simulations is:

- Execute the first simulation program (perhaps multiple times).
- Provide its output files along with other input to another simulation program, executing it (perhaps multiple times).
- Repeat the process.

The primary drawbacks are that:

- Revision of parameters is done by modifying input file elements.
- History can be maintained only at the file/program level.

[UH/ISE Home Page | Phase Two Report Index]



Domain-Level Integration

Jump To: [Previous Page | Next Page]

The domain-model approach to integration has several key benefits in terms of *executing* simulations.

- It provides a conceptual history across different types of actions simulated by *different* programs (effectively extending the benefits we received from applying the domain-model approach to a single program).

  Example: The user can execute a set of Detonations followed by a simulation thatuses the result of one of the detonations (e.g., Compute-Collision-Probabilities) and have a complete record of all action inputs and outputs in terms of domain objects.

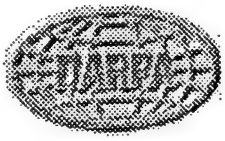
- It allows simple modification of needed input values of later actions in a series of domain actions.

  Example: After executing a Detonation to generate many of the initial values for Compute-Collision-Probabilities (Compute-CP) the user then repeatedly executes the Compute-CP with alternate Space-Object catalogs (but using the same Debris-Clouds produced by the detonation).

Its costs are:

- The execution cost of mapping to and from intermediate objects instead of directly using the files.
- The development cost of producing the domain model and the mapping into and out of it.

[UH/ISE Home Page | Phase Two Report Index]



Supporting Domain-Level Integration With Views

Jump To: [Previous Page | Next Page]

As a result of integration, we may have different domain actions use or produce different pieces of domain objects.

> Example: One action uses the summary information in the Debris-Cloud produced from Break-Up Event, while another uses the detailed information about Fragments.

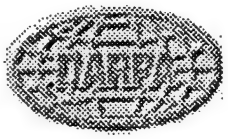
The problems are:

- When obtaining information about a domain object from the user, we need only acquire that info needed by a particular domain action.
- Actions may only produce part of the fields in a domain object, with the rest filled in by the user or other actions.

We address these problems by adding a mapping mechanism called views.

- A view is a domain object that corresponds to a subset of the attributes of another domain object.
- Domain actions can specify their input and outputs in terms of views, when they aren't interested in the complete domain object.

[UH/ISE Home Page | Phase Two Report Index]



How Views Work

Jump To: [Previous Page | Next Page]

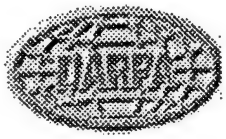
A view is defined by specifying a name for the view and a set of attributes of an underlying domain object.

When a view appears as an input for a domain action, the user can do one of several things:

- *Supply the underlying domain object.* In this case, only the necessary attributes of that object are used. If any view attributes have not been supplied in that underlying object, the user must supply those values.
- *Create an instance of the view and supply values for its attributes.* In this case, the system creates the underlying domain object and fills in only those attributes of it.
- *Supply another instance of the same view.* In this case, the value of that view's attributes are used.

When a view appears as the output of a domain action, the system automatically creates the underlying object and fills in the appropriate attributes.

[UH/ISE Home Page | Phase Two Report Index]



Supporting Simulation Program Evolution

Jump To: [Previous Page | Next Page]

Having an explicit domain model also supports the *evolution* of a collection of related simulations.

Two ways collections of legacy simulations evolve are:

- Adding new tools to work with pieces of the domain model (e.g., adding a new 3-D viewer showing how a single Debris-Cloud forms).
- Replacing an old model with a new model (e.g., replacing the underyling Debris program with a new program that consists of a high-facility orbital modeler, but that requires different parameters).

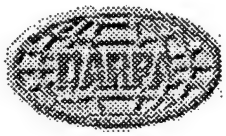
Additions and modifications can be done by modifying or adding input/output mappings and by updating the domain model, without modifying the simulations themselves or their various underlying file formats.

Here are examples of each task and how the domain model facilitates it:

- Adding a new simulation tool
- Replacing an existing simulation tool with a new tool that has an identical file or domain interface
- Replacing an existing simulation tool with a new tool that has a modified domain interface

Essentially, our approach describes simulation programs as if they were modules in an object-oriented programming environment, which gives rise to many of the same benefits.

[UH/ISE Home Page | Phase Two Report Index]



Example: Adding A New Simulation Tool

Jump To: [Next Page]

Adding a new tool (e.g., 3-D viewer for a single Debris-Cloud) generally requires:

- Updating the domain model.

```
View-Space-Object
  |
  +--View-Debris-Cloud
        |  Has-Input
        |     Debris-Cloud
        |
        +--3-D View-Debris-Cloud
              Has-Input
                 Viewer-Location  [Orbital-Location]
```

  Currently, we don't explicitly represent side effects, such as putting an object on the display.

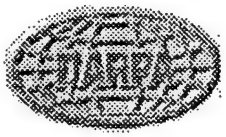
- Adding input/output mappings for the action.

  In this case, the mapping is from a Debris-Cloud and a Viewer-Location to the underlying format the viewing tool requires.

  For viewing actions, we need only worry about the input.

The result is that the viewing tool is isolated somewhat from the details of the files *Impact* produces.

[UH/ISE Home Page | Phase Two Report Index | Evolution Support - Main Page]



Example: Replacing With A "Similar" Simulation Tool

Jump To: [Previous Page | Next Page]

A new tool is *similar* to an old tool in one of two ways:

- *Identical file/command interface*: the replacement differs internally but not externally.

  This situation is the trivial case, and occurs when the new program was designed from internal improvements over the original, with the explicit goal of keeping the interface the same.

  It requires only that the domain action(s) dependent on that program be linked to the new program.

  Example: Replacing the Impact program with an improved program. It requires updating the links between each of the Breakup-Events that Impact simulates to refer to the new Impact program.

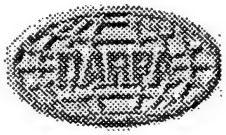
- *Identical domain model interface*: the replacement's I/O differs but it supports the same domain model.

  This situation occurs when the new program finds a different input or output format more convenient for the implementation.

  It requires changing both the links and the mappings between domain actions and the underlying simulation program.

  Example: Replacing the Impact program with an improved program that allows the input information to be specified with less redundancy and uses different units, and that produces its output in a more compact form.

[UH/ISE Home Page | Phase Two Report Index | Evolution Support - Main Page]



Example: Replacing With A "Different" Simulation Tool

Jump To: [Previous Page]

It's more difficult to replace a simulation tool with a different domain interface. That situation tends to occur when we substitute a more sophisticated or more optimized simulation model.

There are several cases we can readily support when substituting a new simulation model:

- *It may require or produce new domain objects or simulate new domain actions.* This case is handled by adding the new objects and actions to the model and providing the appropriate I/O mappings. Existing domain actions and objects are unaffected by this change.

  Example: Replacing *Impact* with a program that simulates several additional types of Breakup-Events.

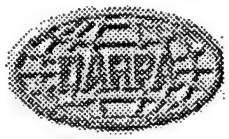
- *It requires or produces objects with fewer attributes.* This case is handled by establishing new views for the objects with fewer attributes and updating the I/O mappings.

  Example: Replacing *Debris* with a program that does not require the summary information produced by Impact and that does not produce summary information in the resulting description of Debris-Clouds.

- *It requires or produces domain objects with additional attributes.* This case can be handled either by adding the new attributes to the existing domain objects or by changing the existing domain objects to views of a more complex underlying object that includes these additional attributes. In either case, the existing I/O mappings must be updated.

  Example: Replacing *Impact* with a program that requires more information about the materials used in the objects and produces more information about each fragment.

[UH/ISE Home Page | Phase Two Report Index | Evolution Support - Main Page]



Additional Integration Support

Jump To ious Page}

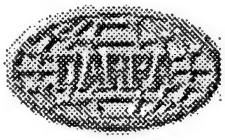
Our mod provides support for integrating programs not designed originally to work together.

Ex Adding a new orbital modeler that requires a Debris-Cloud and
Sp ject-Catalog that are substantially different from those used by other programs.

This pr in be addressed by:

- C lomain objects for the new program, taking advantage of existing objects wherever po
- Pr transforming domain actions that take objects created by existing action and additional inf on and produce the new domain objects.

[UH/ISE Page | Phase Two Report Index]



Project Status

We divide our status report into three parts:

- Project tasks we have successfully completed.
- Project tasks we did not successfully complete.
- Areas we want to explore in the future.

[UH/ISE Home Page]

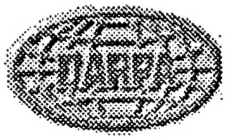
Completed Project Tasks

Jump To: [Next Page]

We have completed the following tasks.

- Exploring in detail the key issues involved in using a domain model as an interface to accessing and executing legacy simulation software.
- Specifying the features such a domain-model environment should provide and how they can be provided.
- Formulating a domain model for a small set of programs used to simulate the formation and orbital behavior of space debris.
- Designing and implementing a simplified version of the domain-driven execution model.

[UH/ISE Home Page | Project Status - Main Page]



Project Tasks Not Completed

Jump To: [Previous Page | Next Page]

We have not completed the following tasks.

- Implementing the entire simulation environment architecture that resulted from our initial exploration.

  Completing an initial implementation is a current masters thesis/project.

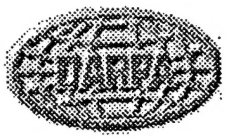
- Providing a complete domain model for a domain consisting of a sizeable set of simulation programs.

  Tackling this domain model is another current masters thesis/project.

- Evaluating the simulation environment with this domain model.

  This task will be addressed upon completion of the implementation and domain model.

[UH/ISE Home Page | Project Status - Main Page]



Future Research

Jump To: [Previous Page]

Some key research issues we will explore in the future:

- Allowing multiple variants of existing models (e.g., adding another possible mechanism for computing collision probabilities, such as a quick and dirty approximation rather than a more computationally intensive but more precise model).

  This requires describing models in terms of more than just domain actions.

- Exploring the benefits of using subsumption to represent the domain model (e.g., allowing users to retrieve simulation actions by properties of the action, rather than by name).

  This requires representing the domain model more formally than in our simple frame-based representation.

[UH/ISE Home Page | Project Status - Main Page]

DISTRIBUTION LIST

AUL/LSE **Bldg 1405 - 600 Chennault Circle** **Maxwell AFB, AL 36112-6424**	**1 cy**
DTIC/OCP **8725 John J. Kingman Rd, Suite 0944** **Ft Belvoir, VA 22060-6218**	**2 cys**
AFSAA/SAI **1580 Air Force Pentagon** **Washington, DC 20330-1580**	**1 cy**
AFRL/PSOTL **Kirtland AFB, NM 87117-5776**	**2 cys**
AFRL/PSOTH **Kirtland AFB, NM 87117-5776**	**1 cy**
University of Hawaii at Manoa **Department of Electrical Engineering** **Holmes Hall 483** **2540 Dole Street** **Honolulu, HI 96822**	**1 cy**
Official Record Copy **AFRL/VSS/Dr. deJonckheere** **Kirtland AFB, NM 87117-5776**	**2 cys**
AFRL/VS **Dr. Fender** **Kirtland AFB, NM 87117-5776**	**1 cy**